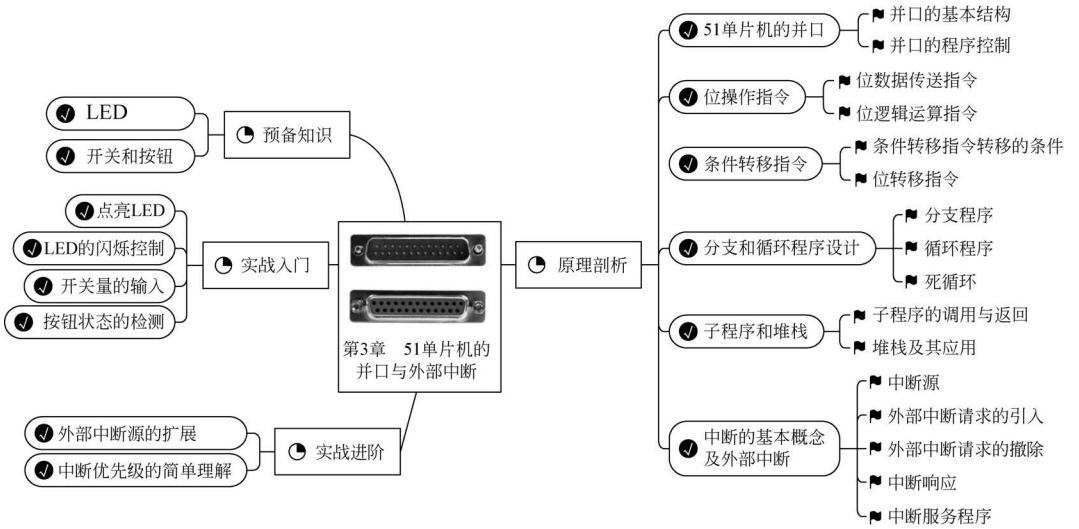
第 3 章

51 单片机的并口与外部中断



并行 I/O 接口(Parallel Input/Output Interface)简称**并口**(PIO),是所有单片机中集成的一项重要内部资源。不同型号的单片机,集成的并口数量有区别,功能上也各有不同,但最基本的功能都是实现多位二进制数据的并行传输。大多数并口也能根据需要只利用一根线单独传输一位二进制代码(这样的代码通常称为**开关量**),从而实现对外部各种开关设备的状态检测和控制。

在与这些外部设备之间进行数据传送过程中,广泛采用中断技术。**中断**(Interrupt)是现代各种微机系统中广泛采用的一项基本技术。对 51 单片机来说,通过特定的并口引脚可以将外部设备的状态信息作为中断事件引入 51 单片机,这些中断称为**外部中断**。本章也将结合并口引入的外部中断,介绍 51 单片机中断系统的一些基本概念。

3.1 磨刀霍霍——预备知识

发光二极管(Light-Emitting Diode,**LED**)是一种将电能转化为光能的半导体电子元件。这种电子元件早在 1962 年出现,早期只能发出低亮度的红光,之后发展出其他单色光

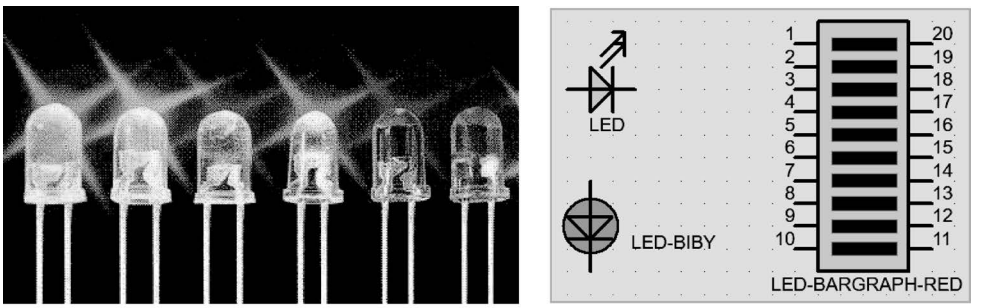
的版本，时至今日能发出的光已遍及可见光、红外线及紫外线，亮度也有极大提高。LED早期只是作为指示灯、显示板等，随着技术的不断进步，目前已被广泛地应用于显示器、电视机、采光装饰和照明。

LED是典型的开关量**输出设备**(Output Device)，只能接收从单片机并口输出的数据，实现其亮/灭状态的控制。实际系统中，还有像**开关**(Switch)和**按钮**(Button)之类的**输入设备**(Input Device)，这些外部设备的作用是产生数据或者控制信号送入单片机，对单片机的工作进行控制。

3.1.1 LED

图3-1给出了LED的外观及其在Proteus中的电路符号。其中，在一个LED-BARGRAPH-RED器件中同时集成了10个LED，引脚1～10为阳极，引脚11～19为阴极。

图3-1



(a) 外观　　(b) 电路符号

图3-1　LED外观及电路符号

图3-2为LED的简单控制电路示例。对共阳极接法来说，当通过外部电路使IO端为低电平时，LED上有正向压降。当IO端的低电平足够小，或者限流电阻R2足够小时，流过的正向电流足够大，则LED点亮。

图3-2

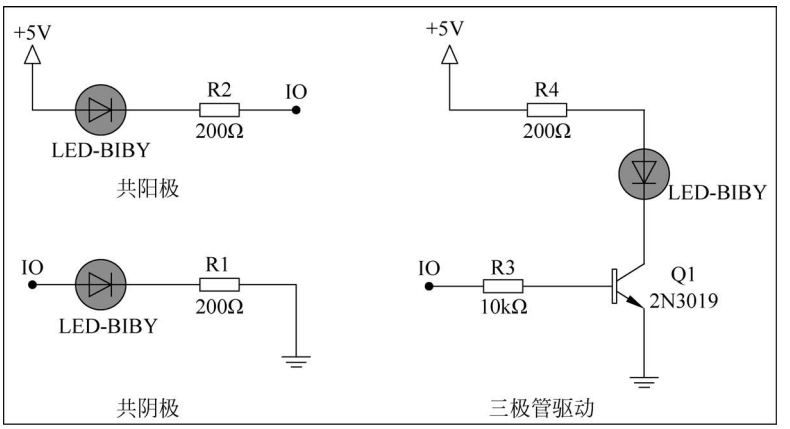


图3-2　LED的简单控制电路

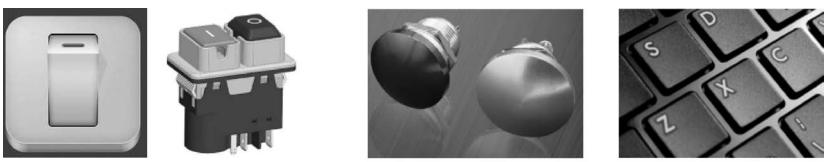
同理，对共阴极接法来说，当通过外部电路使 IO 端为高电平时，LED 上有正向压降。当 IO 端的高电平足够大，或者限流电阻 R2 足够小时，流过的正向电流足够大，则 LED 点亮。

LED 的一般工作电压为 1.8～2.2 V，工作电流为 1～20 mA。流过的电流越大，亮度越高。但是，如果流过的电流太大，LED 可能被烧毁。因此实际电路中都必须加**限流电阻** R1 和 R2。

即便有正向电流流过，如果电流太小，亮度也不能满足用户要求。为此，可以利用三极管等实现电流的放大。图 3-2 中，外部控制信号通过 IO 端送入三极管的基极。当控制信号为高电平使三极管导通时，集电极有比较大的电流流过，从而控制 LED 点亮。

3.1.2　开关和按钮

开关和按钮的典型外观如图 3-3 所示。这两种外设都属于开关量输入设备，利用这些设备可以对单片机系统的运行过程进行控制，或者在运行过程中向单片机实时输入一些简单的命令等。

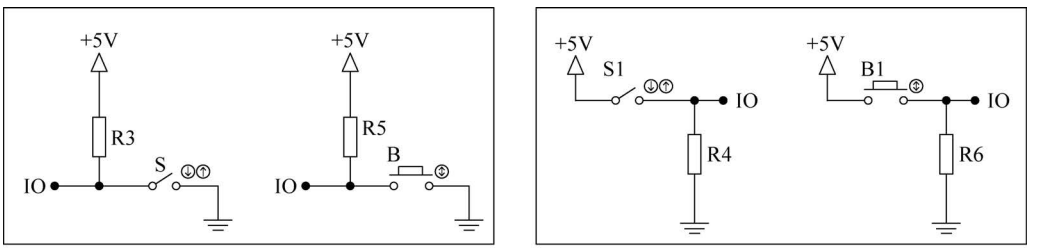


(a) 开关　　(b) 按钮

图 3-3　开关和按钮的典型外观

图 3-3

开关和按钮都需要相应的电路将触点的通断状态转换为高/低电平，才能送入单片机。典型的连接电路如图 3-4 所示，图中 S 代表开关，B 代表按钮。在图 3-4(a)中，当开关或按钮按下时，IO 端为低电平；开关断开或按钮释放时，IO 端为高电平，R3 和 R5 称为**上拉电阻**。图 3-4(b)所示电路正好相反，R4 和 R6 称为**下拉电阻**。



(a) 共阴极接法　　(b) 共阳极接法

图 3-4　开关和按钮电路

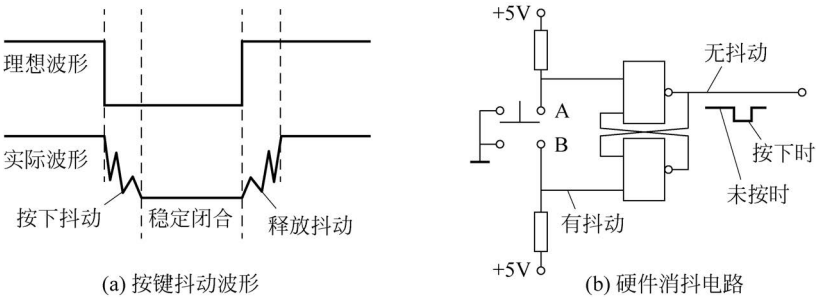
开关和按钮的功能和具体应用存在一定的区别。在系统运行过程中，用户通过手指按下开关或按钮，开关或按钮的触点闭合。当松开手指后，开关的触点将一直保持闭合状态。

如果需要断开，需要重新扳动开关。但是对按钮来说，松开后，按钮触点又自动回到原来的通断状态。

在上述电路中，不管是开关还是按钮都有一个共同的问题，即按键抖动。以共阴接法的按钮电路为例，在按钮按下和释放的过程中，IO 端电平的变化波形如图 3-5(a)所示。

正常情况下，按钮每按下一次，IO 端输出一个低电平脉冲。由于按键抖动，在按下和释放的过程中，IO 端可能输出多个低电平脉冲，称为**抖动**。

实际系统中必须采取措施消除抖动的影响，常用的有程序消抖法和硬件电路消抖法。图 3-5(b)给出了一个简单的硬件消抖电路，其中利用 RS 触发器的工作原理实现硬件消抖。



(a) 按键抖动波形　　(b) 硬件消抖电路

图 3-5　按键抖动波形和硬件消抖电路

微课视频

3.2　小试牛刀——实战入门

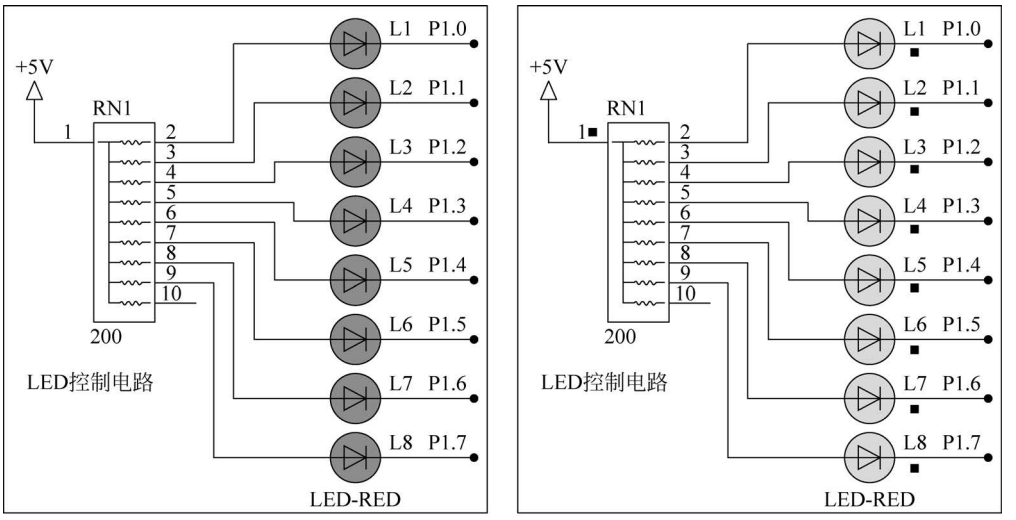
本节先通过几个案例了解 LED、开关和按钮的基本工作原理、电路连接及 51 单片机对其进行控制的基本方法。

动手实践 3-1：点亮 LED

本案例的控制电路原理图如图 3-6 所示(参见文件 **ex3_1. pdsprj**)。图中 RN1 为电阻排，其中集成了 9 个电阻，分别用作各 LED 的限流电阻。各 LED 采用共阳极接法，其阳极分别通过各自的限流电阻接＋5V 电源，而阴极分别与单片机的 P1.0～P1.7 引脚相连接，当某个引脚输出低电平时，对应的 LED 点亮。

(1) 要使 8 个 LED 全部点亮，可以编写如下完整的汇编语言程序(参见文件 **p3_1_1. asm**)：

```
; 点亮 8 个 LED 灯
LED   EQU  P1          ; 将控制 LED 的 P1 口定义为符号常量 LED
      ORG  0000H
      AJMP MAIN
      ORG  0100H
MAIN:MOV   LED,#00H  ; 点亮 8 个 LED
      END
```



(a) LED全部熄灭的状态　　(b) LED全部点亮的状态

图 3-6

图 3-6　51 单片机控制 LED 的亮/灭的电路

在程序中，首先用伪指令 EQU 将控制 LED 的 P1 并口定义为符号常量 LED。在主程序中，将符号常量 LED 作为 MOV 指令的目的操作数，即代表 P1 口。执行 MOV 指令后，将立即数 00H 传送到 P1 口对应的内部 RAM 单元，从而控制由 51 单片机 P1 并口对应的 8 个引脚输出低电平，点亮 LED。

（2）如果希望只点亮一个 LED（如图 3-7 所示），可以将上述主程序中的 MOV 指令替换为如下指令（参见文件 **p3_1_2.asm**）：

```
MAIN:   MOV   LED,#11110111B
```

或者

```
MAIN:   CLR    L4
```

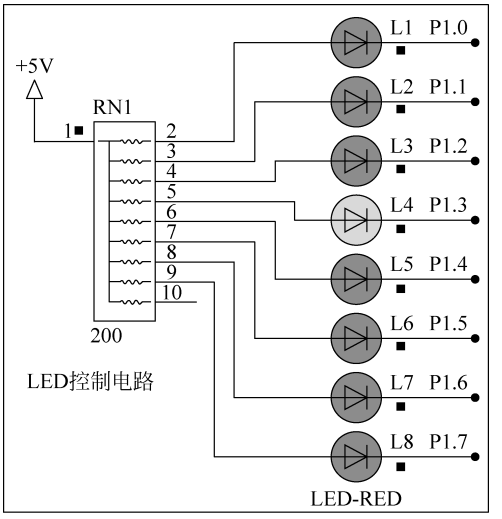


图 3-7　只点亮 L4 的运行效果

其中 L4 为在程序一开始用如下伪指令定义的位变量：

```
L4     BIT     P1.3
```

动手实践 3-2：LED 的闪烁控制

通过程序可以控制 LED 以一定的时间间隔闪烁。假设要求图 3-6 中的 L4 灯以 1 s 左右的时间间隔闪烁，可以编写如下汇编语言程序（参见文件 **p3_2.asm**）：

```
; LED的闪烁控制(用子程序实现延时)
L4     BIT    P1.3          ; 将控制 L4 的 P1.3 引脚定义为位变量 L4
       ORG    0000H
       AJMP   MAIN
;=============================================================
; 主程序
;=============================================================
       ORG    0100H
MAIN:  MOV    SP,#60H    ;设置堆栈指针
       CLR    L4
       ACALL  DELAY      ; 调用延时子程序
       SETB   L4
       ACALL  DELAY      ; 调用延时子程序
       SJMP   MAIN
;=============================================================
; 延时子程序
;=============================================================
DELAY:MOV     R5,#13
LP3:   MOV    R7,#200      ; 1 个机器周期
LP4:   MOV    R6,#200      ; 1 个机器周期
       DJNZ   R6,$         ; 2 个机器周期
       DJNZ   R7,LP4       ; 2 个机器周期
       DJNZ   R5,LP3
       RET
       END
```

主程序一开始，首先设置堆栈指针，并用 CLR 指令使 P1.3 引脚输出低电平，从而点亮 L4 灯。之后调用延时子程序 DELAY 实现近似 1 s 延时，再执行 SETB 指令将 L4 灯熄灭，延时近似 1 s 后又跳转回 MAIN 语句，重复上述过程。

动手实践 3-3：开关量的输入

开关量的输入电路原理图如图 3-8 所示（参见工程文件 **ex3_3. pdsprj**）。其中 DSW1 为开关排，内部集成了 8 个开关。U2 为 LED 排，内部集成了 10 个 LED，本案例只用了其中的 8 个 LED。RN1 为电阻排，设置内部 9 个电阻的参数都为 200Ω，电路中只将其中的 8 个电阻分别作为 8 个 LED 的限流电阻。

图 3-8

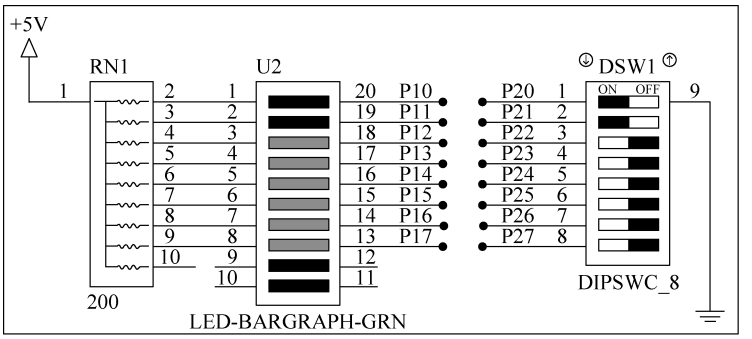


图 3-8　开关量的输入电路原理图

根据上述电路连接，编写汇编语言程序实现如下功能：用 8 个 LED 指示 8 个开关的通断状态。例如，当开关 1、2 打在 ON 位置，其余开关打在 OFF 位置时，对应最上面两个 LED 熄灭，其他的 LED 点亮。完整的代码如下（参见文件 **p3_3.asm**）：

```
; 开关数据的输入
LED     EQU     P1              ; 将 LED 控制口 P1 定义为符号常量 LED
SW      EQU     P2              ; 将开关输入口 P2 定义为符号常量 SW
        ORG     0000H
        LJMP    MAIN
        ORG     0100H
MAIN:   MOV     SW,#0FFH        ; P2 口先输出高电平
LP:     MOV     A,SW            ; 读入开关数据
        CPL     A               ; 取反
        MOV     LED,A           ; 输出控制 LED
        SJMP    LP              ; 循环
        END
```


微课视频

在电路中，8 个开关的通断状态被转换为 8 个高/低电平，可以视为一字节的二进制数据，由符号常量 SW 代表的 P2 口送入单片机。在程序中，利用第二条指令将开关数据送入累加器 A，将其取反后通过由符号常量 LED 代表的 P1 口输出控制 LED 的亮灭。


微课视频

动手实践 3-4：按钮状态的检测

按钮状态的检测电路原理图如图 3-9 所示（参见 Proteus 工程文件 **ex3_4.pdsprj**）。图中按钮电路的输出接到 P3.2 引脚。每按动一次按钮，将累加器 A 中的内容加 1，同时 LED 灯 L1 闪烁一次（亮/灭切换一次）。

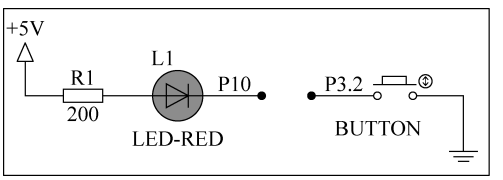


图 3-9　按钮状态的检测


图 3-9

（1）采用查询方式检测按钮状态。

实现上述功能的代码如下（参见文件 **p3_4_1.asm**）：

```
; 按钮状态的检测——查询方式
BT      BIT     P3.2            ; 将按钮状态输入引脚 P3.2 定义为位变量 BT
LED     EQU     P1              ; 将控制 LED 的 P1 口定义为符号常量 LED
        ORG     0000H
        AJMP    MAIN
        ORG     0100H
MAIN:   SETB    BT              ; BT 先输出高电平
        CLR     A               ; 按动次数初始化为 0
LP:     JB      BT,$            ; 等待按钮按下
        JNB     BT,$            ; 等待按钮释放
        INC     A               ; 按动次数加 1
        XRL     LED,#01H        ; L1 灯亮/灭切换，闪烁一次
        SJMP    LP              ; 死循环
        END
```

在上述程序中，首先利用位操作指令 SETB 使位变量 BT 代表的单片机的 P3.2 引脚输

出高电平，之后将累加器 A 中的内容清零，作为统计按钮按动次数的初始值。

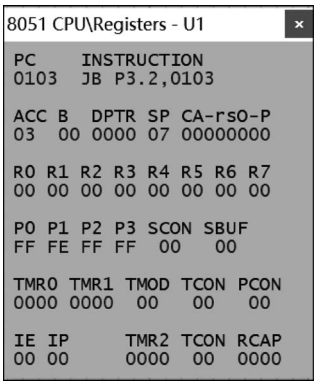


图 3-10 寄存器观察窗口

在程序后面的循环中，重复检测按钮是否按下和释放。检测到按钮每按动一次，将按动次数加 1，并利用逻辑运算指令 XRL 控制 L1 灯亮/灭切换一次（闪烁一次）。

将程序编译生成 HEX 文件后，在原理图中加载并启动运行。在运行过程中单击原理图中的按钮，LED 将不断闪烁。在暂停运行状态下，将自动弹出单片机的寄存器观察窗口，在其中可以观察到累加器 A 中存放的按钮按动次数，如图 3-10 所示。

（2）采用中断方式检测按钮的状态。

如果要求采用中断方式实现按钮状态的检测，则重新编制的完整汇编语言程序如下（参见文件 **p3_4_2.asm**）：

```
; 按钮状态的检测——中断方式
BT     BIT    P3.2          ; 定义位变量 BT
LED    EQU    P1            ; 定义符号常量 LED
       ORG    0000H
       AJMP   MAIN
       ORG    0003H
       AJMP   IDEL          ; 中断服务程序入口
; ============================  =================
; 主程序
; ================================================
       ORG    0100H
MAIN:  MOV    SP,#60H       ; 设置堆栈指针
       SETB   BT            ; P3.2 先输出高电平
       CLR    A             ; 按动次数初始化为 0
       SETB   IT0           ; 设置外部中断 0 边沿触发方式
       SETB   EX0           ; 开中断
       SETB   EA
       SJMP   $             ; 等待中断
; ============================================================
; 中断服务子程序
; ============================================================
IDEL:  INC    A             ; 按动次数加 1
       XRL    LED,#01H      ; L1 灯闪烁
       RETI   ; 中断返回
       END
```

上述程序主要包括主程序和中断服务子程序，其中涉及的很多概念将在后面详细介绍。

3.3 庖丁解牛——原理剖析

在上述各案例中，主要涉及 51 单片机的并口及数据的输入/输出、位操作指令和条件转移指令的用法、子程序的调用与返回、汇编语言中循环程序和外部中断的概念及中断服务子

程序的编写方法。下面对这些内容逐一进行介绍。

3.3.1　51 单片机的并口

51 单片机内部集成了 4 个并口 P0～P3。在 51 单片机引脚上，每个并口对应一组 8 个引脚，共有 4 组引脚，可以并行传送 8 位二进制数据。以 P0 口为例，对应的 8 个引脚为 P0.0～P0.7。

1. 并口的基本结构

51 单片机每个并口内部都由完全相同的 8 套电路构成，而不同并口的内部电路结构有一些细微区别，从而导致在功能和用法上有所不同。

(1) P1 口。

图 3-11 是 P1 口内部的一位电路结构。其中 P1.x(x=0～7)代表 P1 口的一位引脚，通过这些引脚可以连接外部的按钮、开关和 LED 等。这些引脚通过并口电路中的“读锁存器”“写锁存器”“读引脚”命令信号线和内部总线与 51 单片机内部的 CPU 等其他部件相连接。

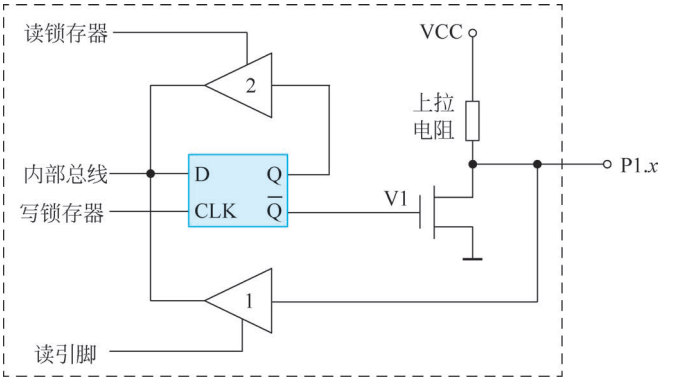


图 3-11　P1 口内部的一位电路结构

通过并口可以实现数据的并行输入(Input)或输出(Output)。当工作在输出方式时，51 单片机中的 CPU 通过执行数据输出指令，将指令中给定的 0 码或 1 码送到内部总线，同时由 CPU 发出“写锁存器”命令。在此命令作用下，D 触发器输出的 $\overline{Q}$ 端输出对应的高/低电平，从而控制场效应管 V1 的断开和接通，使引脚 P1.x 分别得到低电平或高电平，再进一步送到 LED 等外部设备。

当工作在输入方式时，CPU 通过执行输入数据的指令，送来“读引脚”命令。在此命令作用下，将三态门 1 打开，从而将 P1.x 引脚与内部总线接通，51 单片机中的 CPU 即可检测到引脚上由外部电路(例如按钮电路)决定的电平状态，从而读入一位二进制数据。

对同一个并口的 8 套电路，其中的 8 个 D 触发器构成锁存器(Latch)，8 个三态门构成缓冲器(Buffer)。只要 CPU 不执行新的数据输出命令，锁存器的输出高/低电平就不会改变，从而使得 8 个引脚稳定地输出前面送出的 8 位数据以及对应的高/低电平。同理，外部

电路在不断工作，使得 8 个并口引脚的高/低电平在不断变化。但是，只要 CPU 不执行新的数据输入命令，缓冲器就不会打开，并口引脚的高/低电平以及对应的数据就不会送到内部总线和 CPU。

（2）P0 口。

P0 口内部的一位电路结构如图 3-12 所示，与 P1 口的区别主要有如下两点：

- P0 口内部没有上拉电阻，而是替换为场效应管 V2；
- P0 口电路中增加了一个“控制”输入端和相应的控制电路。

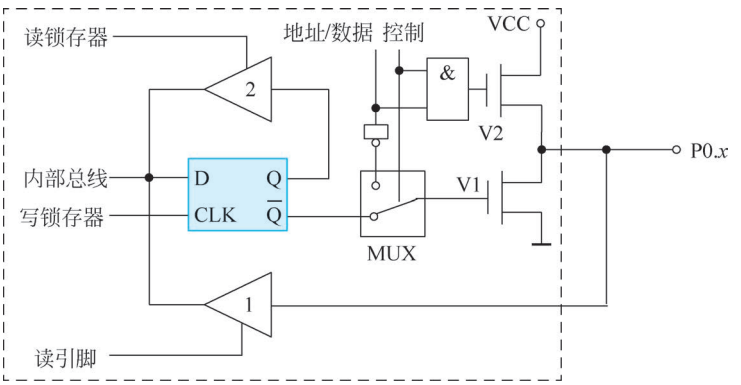


图 3-12　P0 口内部的一位电路结构

当 P0 口用作普通的并行输入/输出接口时，CPU 内部使得“控制”端为低电平，从而使开关 MUX 打到下方，同时通过与门使 V2 始终处于断开状态，V1 的漏极始终处于开路状态，称为**漏极开路**。

在这种情况下，如果需要并口引脚上输出 1 码，则与 P1 口一样，首先使 D 触发器 $\overline{Q}$ 端输出低电平，并控制 V1 截止。此时由于 V1 和 V2 同时截止，因此引脚 P0.x 将处于高阻悬空状态，输出高/低电平不确定。如果通过该引脚连接 LED，则无法使 LED 上获得确定的正向压降，从而不能保证 LED 一定熄灭或点亮。为此，必须在外部电路上采取相应的措施。

当“控制”端为高电平时，P0 口不是用作普通的并口，而是用于传送 51 单片机外部存储器单元的地址和数据。这一点将在后面相关章节再做介绍。

（3）P2 口。

P2 口内部的一位电路结构如图 3-13 所示。与 P1 口相比，只是在内部增加了一个开关 MUX。在“控制”端信号作用下，当开关 MUX 与下侧触点接通时，可以实现与 P1 口完全一样的普通并口功能。当开关打在上方触点时，与 P0 口类似，P2 口不是用作普通的并口，而是用于传送外部存储器单元的地址。

（4）P3 口。

P3 口内部的一位电路结构如图 3-14 所示。当图中的“第二功能输出”端为高电平时，电路结构与 P1 口完全一样，用作普通并口。

在 51 单片机系统中，P3 口很多情况不是用于普通的并口，而是实现特定的其他功能

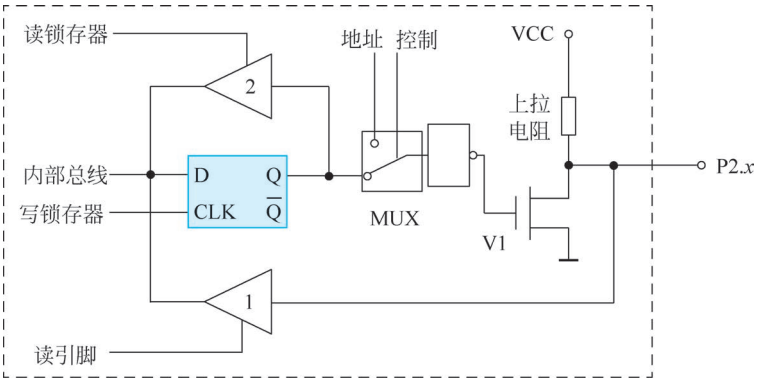


图 3-13　P2 口内部的一位电路结构

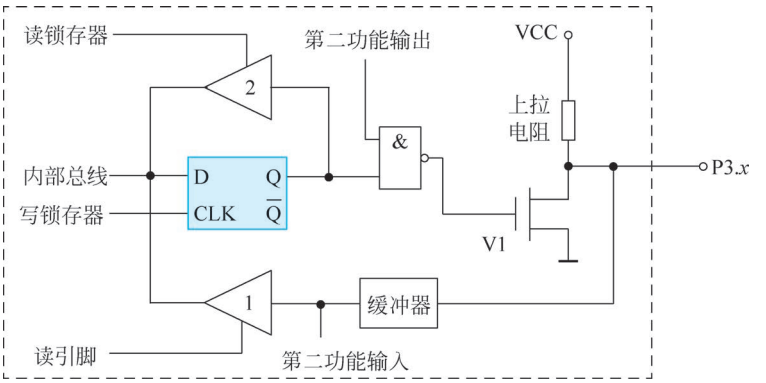


图 3-14　P3 口内部的一位电路结构

(称为并口的**第二功能**),例如用作串口数据线、外部存储器读写控制信号线。此时,这些功能命令对应的高/低电平通过与非门直接控制 V1,使相应的引脚输出或输入第二功能命令信号。具体每个引脚的第二功能将在后面相关章节陆续介绍。

根据上述电路结构,下面对 P0～P3 用作普通并口时的基本用法做一个简单总结。

- 在 4 个并口中,P0 口是**双向口**,通过 P0 口的地址或数据可以是由 CPU 输出到外部,也可以是由外部输入 CPU。另外的 3 个并口是**准双向口**,在一个应用系统中,要么连接输出设备实现数据的输出,要么连接输入设备实现数据的输入。
- 4 个并口内部都有**锁存器**,能够锁存输出数据,以便稳定地驱动外部电路和设备的工作。并口内部也都有**缓冲器**,实现输入数据时的缓冲,也就是能够由 CPU 决定在适当的时刻从并口引脚读入期望的数据。
- P0 口用作普通的输出并口时,必须外接**上拉电阻**,以便使并口引脚上输出确定的高/低电平。
- 从任何一个并口输入数据时,必须先使该并口输出高电平,以便使并口内部的 V1 场效应管截止断开,引脚上的高低电平能够正确反映外部电路和设备的状态。

2. 并口的程序控制

在51单片机外部，并口通过相应的引脚与外部设备和电路相连接。而在51单片机内部，并口P0～P3分别对应4个特殊功能寄存器，地址分别为80H、90H、0A0H和0B0H。通过用MOV指令将一个8位二进制数据传送到特殊功能寄存器，即可实现51单片机与并口（从而与外部设备）之间的数据传送，并进一步通过引脚对所连接的外部设备进行访问控制。

例如，在**动手实践 3-1**中，8个LED分别连接到51单片机P1口的8个引脚P1.0～P1.7。当执行如下数据输出指令时，将立即数00H传送到符号常量LED代表的P1口，并通过P1口内部电路使P1.0～P1.7引脚全部输出低电平，从而控制8个LED全部点亮。

```
MOV  LED,#00H
```

在**程序 p3_1_2.asm**中，将指令的操作数00H替换为0F7H＝11110111B，由于其中只有D3＝0，此时只有P1.3引脚输出低电平，从而只点亮L4灯。

在**动手实践 3-3**中，开关排的8个开关分别连接P2口，另一端都接地。因此开关的通/断将使P2口对应的引脚为高/低电平。每个引脚上的高/低电平分别用1码和0码表示，从而得到8位二进制数据，利用如下数据输入指令：

```
MOV  A,SW
```

将其从P2口读入51单片机。由于开关是输入设备，因此P2口在本案例中用作输入口，在主程序的一开始，用如下语句：

```
MAIN:  MOV  SW,#0FFH
```

使符号常量SW代表的P2口的8个引脚先输出高电平，以便后面正确检测到开关的通/断状态。

3.3.2 位操作指令

在**动手实践 3-1**和**3-2**中，只需要点亮通过并口的某个引脚连接的一个LED，或者让指定的L4灯闪烁，可以用位操作指令对并口中该引脚对应的二进制位进行单独控制，而其他位不受影响。这样的操作称为**位操作**。

在51单片机中，专门提供了一类指令实现位操作，具体包括位数据传送、位逻辑运算和位转移三类指令。位操作指令是51单片机中功能十分强大、使用非常频繁和灵活的指令。这里首先介绍前面两类指令。

1. 位数据传送指令

位数据传送指令实现在PSW中的进位标志位C与普通位单元之间的一位二进制数据传输，指令的基本格式为：

```
MOV  C,bit
MOV  bit,C
```

例如，如下指令：

```
MOV  C,00H
```

将位地址为00H的位单元(内部RAM中20H单元的最低位)中原来保存的一位二进制数1码或0码传送到C标志位。

这两条指令中的操作数bit可以是内部RAM中位寻址区中的位单元，也可以是特殊功能寄存器区中能够进行位寻址的某个特殊功能寄存器的指定位。在指令中一般直接写出其位地址，称为**位寻址**。位于特殊功能寄存器区中能够位寻址的位单元既有位地址，也有位名称，在指令中一般直接用其名称表示。例如，

```
MOV  C,P1.0
```

其中的P1.0是特殊功能寄存器P1口的D0位，也代表P1口的最低位引脚。该条指令实现的功能就是将P1.0引脚上用高/低电平表示的一位1码或0码读入CPU，并存入C标志位。

2. 位逻辑运算指令

位逻辑运算指令实现对一位二进制操作数的清0(**复位**)、置1(**置位**)和位逻辑运算(**位与、位或、位取反**)。

(1) 位单元的复位和置位操作。

位单元的复位和置位操作分别用指令CLR和SETB实现，实现的基本功能是向指定的位单元分别写入一位二进制代码0或1。如果位单元是指定并口的某一位，则当向其中写入0码或1码时，相应的引脚将输出低电平或高电平。

例如，在动手实践3-1的**程序 p3_1_2.asm**中，如下指令使符号位地址L4代表的P1.3引脚输出低电平，其中的操作数为P1口的D3位。

```
CLR    L4
```

类似的，如下指令：

```
SETB  P1.7
```

或者

```
SETB  97H
```

将P1.7置1，即通过P1.7引脚输出高电平，其中P1.7代表P1口的D7位，而该位单元的位地址为97H。

在**动手实践 3-4**中，由于按钮是输入设备，因此在主程序的一开始，用如下语句：

```
MAIN:  SETB  BT
```

使符号位地址BT代表的P3.2引脚先输出高电平，以便后面正确检测到按钮的通/断状态。

（2）位逻辑运算。

每个位单元中保存的是一位二进制代码 1 或 0，这些二进制代码之间可以进行逻辑与、或、非运算，在 51 单片机中分别用 ANL、ORL 和 CPL 指令实现。其中逻辑与和逻辑或运算指令需要两个操作数，而逻辑非运算指令只需要一个操作数。

在 ANL 和 ORL 指令中，源操作数可以是任何一个位单元，但目的操作数必须是 C。例如，

```
ANL   C,10H
```

将 PSW 中的最高位与位地址为 10H 的位单元中的一位二进制代码进行与运算，结果放回 C。

此外，ANL 和 ORL 的源操作数还可以是某个位单元的取反。例如，如下指令实现的功能是将位单元 10H 中的一位二进制代码取反后再和 C 标志进行逻辑与运算。

```
ANL   C,/10H
```

在 CPL 指令中，操作数可以是 C 或任意一个位单元，实现的功能是将指定的位单元中保存的一位二进制代码取反，再放回去。

对上述位逻辑运算指令，再做几点说明：

- 在 51 单片机的指令系统中，ANL、ORL 和 CPL 指令可以实现普通的逻辑运算，也可以实现位逻辑运算。这是两大类不同的指令，二者的基本区别在于指令中的操作数是字节操作数还是位操作数。分析和编程时一定要注意是字节操作还是位操作。
- 在普通的字节操作指令（例如 MOV 指令）中，累加器用 A 表示。在位操作指令中，要对累加器 A 中的指定某一位进行位操作，必须用 ACC 而不是 A。例如，如下指令：

```
SETB   ACC.0
```

将累加器 A 中的最低位置为 1，不能写成

```
SETB   A.0
```

3.3.3 条件转移指令

与无条件转移指令相比，条件转移指令也是实现程序执行流程的跳转，但这种跳转是有条件的。只有当给定条件满足后，才跳转到目标指令。如果给定的条件不满足，则不跳转而继续执行转移指令后面的指令。

在 51 单片机的指令系统中，提供了 4 条条件转移指令，即 JZ、JNZ、DJNZ 和 CJNE，这些指令有如下几种书写格式：

```
JZ      rel
JNZ     rel
```

```
DJNZ  Rn,rel
DJNZ  dir,rel
CJNE  A,#data,rel
CJNE  Rn,#data,rel
CJNE  @Ri,#data,rel
CJNE  A,dir,rel
```

在上述指令中，操作数 rel 给定转移的目标单元，即跳转的距离或偏移量。与 SJMP 指令类似，在程序中一般用转移目标指令的标号作为该操作数。指令中的其他操作数可以采用各种寻址方式，例如操作数 Rn 为寄存器寻址；dir 为直接寻址；#data 为立即寻址；@Ri 为寄存器间接寻址。

1. 条件转移指令转移的条件

不同的条件转移指令，其主要区别在于转移的条件不同。

(1) JZ 和 JNZ 指令。

JZ(Jump if Zero)和 JNZ(Jump if Not Zero)指令转移的条件是该指令执行前，累加器 A 中的内容是否为 0(Zero)。对 JZ 指令，当 A=0 时，条件满足，程序跳转到目标指令；对 JNZ 指令则相反，当 A≠0 时跳转。

例如，有如下程序段：

```
      MOV  A,R0
      JZ   ZERO
      XRL  A,#0FH
      SJMP DONE
ZERO: XRL  A,#0F0H
DONE: …
```

在上述程序中，首先将 R0 中的数据存入累加器 A，再利用 JZ 指令检测该数据是否为 0。如果为 0，则条件满足，跳转到标号为 ZERO 的指令，将 A 中的数据与 0F0H 进行逻辑异或运算(也就是将其高 4 位取反，低 4 位保持不变)，再将运算结果存回 A。如果条件不满足，即 A 中的数据不为 0，则不跳转，而继续执行程序中的第一条 XRL 指令，将 A 中的数据低 4 位取反，高 4 位保持不变。

(2) DJNZ 指令。

DJNZ(Decrement and Jump if Not Zero)指令的操作数可以有两种不同的寻址方式，即寄存器寻址和直接寻址。两种情况下，该指令都将依次执行如下两个操作：

- 将操作数减 1，并存回原位置；
- 判断操作数减 1 后是否等于 0，当不等于 0 时跳转；否则顺序执行后面的指令。

例如，如下指令每执行一次，将 R7 中的数据减 1 再存回 R7。

```
DJNZ  R7,LP2
```

之后判断 R7 是否等于 0。如果不为 0，则跳转到标号为 LP2 的指令继续执行；否则不跳

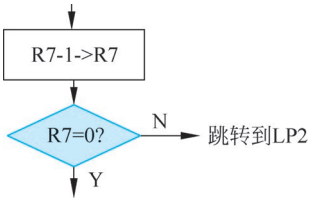


图 3-15　指令“DJNZ R7，LP2”的执行流程

转，继续执行该指令后面的指令。该指令的执行流程可以用图 3-15 表示。

（3）CJNE 指令。

CJNE(Compare and Jump if Not Equal)指令是 51 单片机中唯一一条有 3 个操作数的指令，该指令将前面两个操作数进行比较，并将二者不相等(Not Equal，NE)作为转移的条件。

例如，指令：

```
CJNE   A,＃20H,LP
```

实现的操作和步骤如下：

- 将累加器 A 中的数据与立即数 20H 相减，比较二者是否相等，同时设置 C 标志位；
- 如果两个操作数不相等，则跳转到标号为 LP 的指令继续执行；否则继续执行后面的指令。

需要注意的是，在做两个操作数的相减比较时，将根据减法运算是否有借位设置 C 标志位。但是该减法运算的结果不会保存，因此执行后前面两个操作数的值都不影响。

2．位转移指令

位转移指令是将某个位单元中保存的一位二进制数据是 0 还是 1 作为转移的条件，因此该类指令既属于条件转移指令，也属于位操作指令。

51 单片机中的位转移指令有 5 条，即 JB、JNB、JBC、JC 和 JNC。其中，JB 和 JNB 指令的目的操作数必须是一个位单元，源操作数为跳转的目标指令。两条指令的区别在于：当 JB 指令中目的操作数指定的位单元为 1 时，跳转到源操作数指定的目标指令；而 JNB 指令正好相反，当目的操作数为 0 时跳转到目标指令。

JBC 指令的功能与 JB 指令类似，区别在于：在执行 JBC 指令时，还将目的操作数指令的位单元清 0。此外，与上述 3 条指令不同，JC 和 JNC 指令固定由 C 标志位的状态作为转移的条件，因此不需要另外指定位单元而只有一个操作数。

在程序 **p3_4_1.asm** 中，JB 指令用于检测符号位地址 BT 代表的 P3.2 引脚是否为高电平。如果是，则跳转到这条指令本身，继续读取并检测 P3.2 的状态，直到通过外部的按钮电路将该引脚复位为低电平。因此该条指令的作用是等待按钮按下。

在按钮按下后，JB 指令中的条件 P3.2＝1 不再满足，因此这条指令执行后不再跳转，继续执行下一条 JNB 指令。在执行 JNB 指令时，又不断检测 P3.2 是否为 0。如果按钮按下后一直不松开，则 P3.2 一直为低电平，该指令跳转的条件始终满足，因此将不断重复执行这条指令。一旦按钮松开，则跳转的条件不再满足，从而不再跳转重复，程序继续往下运行。由此可知，在该程序中 JNB 指令的作用是等待按钮松开。

3.3.4　分支和循环程序设计

与高级语言程序类似，汇编语言程序也有两种典型的程序结构，即分支和循环。在高级

语言程序中，利用 if、for 等语句实现分支和循环，而在汇编语言的指令系统中，分支和循环程序主要用上述各种条件和无条件转移指令实现。

1. 分支程序

分支程序包括单分支、双分支、三分支和多分支结构，图 3-16 给出了两种基本的分支结构程序（即单分支和双分支）流程示意图。

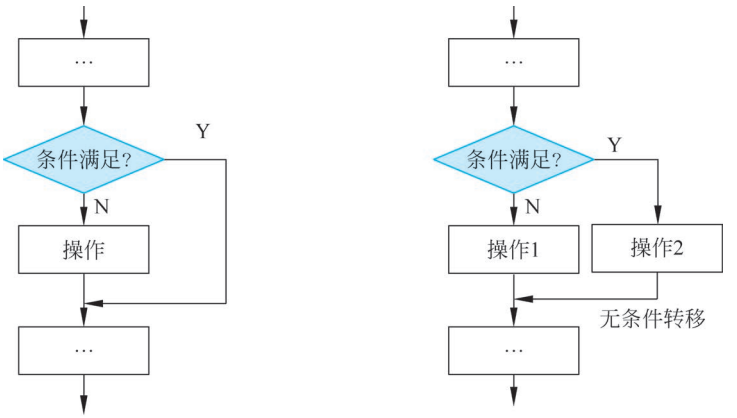


图 3-16　两种基本的分支结构程序流程示意图

（1）单分支结构。

在图 3-16(a)所示的单分支结构中，通过执行条件转移指令实现分支。当跳转的条件不满足时，才执行相应的操作。

例如，要检测累加器 A 中数据的正负，当为负数时将－1 存入寄存器 B，当为 0 和正数时不做任何操作，可以编写如下程序段：

```
        ANL     A,#80H
        JZ      PLUS            ; A 中数据为 0 或正数,则跳转
        MOV     B,#(-1)         ; 否则,将 -1 存入寄存器 B
PLUS:   …
```

在该程序段中，第一条指令将 A 中的数据与 80H 进行逻辑与运算。当 A 中数据为正数或 0 时，其补码的最高位为 0，执行 ANL 指令后 A＝0；当 A 中数据为负数时，其补码的最高位为 1，执行 ANL 指令后 A＝80H≠0。因此，在执行 ANL 指令后，只需要用 JZ 或者 JNZ 指令即可检测 A 中数据的正负。

在上述程序中，用位转移指令 JZ 实现正负数的检测和分支。如果将 JZ 指令替换为 JNZ 指令，为了实现同样的功能和分支，则需要将上述程序段改写为如下：

```
        ANL     A,#80H
        JNZ     MINUS           ; A 中数据为负数,则跳转
        SJMP    CON             ; 否则,跳转
MINUS:  MOV     B,#(-1)         ; 将 -1 存入寄存器 B
CON:    …
```

注意，其中必须增加一条无条件转移指令 SJMP。

（2）双分支结构。

在图 3-16(b)所示双分支结构中，当条件满足或不满足时，分别执行两个不同的操作，实现两路分支。两路分支最后再会合到一起，继续执行程序中后面的操作。

例如，要根据 A 中数据的正负，分别向 R0 中存入＋1 和－1，可以编写如下双分支结构程序：

```
        ANL     A,#80H
        JNZ     MINUS
        MOV     R0,#1          ; A 中原来的数据为非负数,则将 +1 存入 R0
        SJMP    CON
MINUS:  MOV     R0,#0FFH       ; 否则,将 0FFH(即 -1 的补码)存入 R0
CON:    …
```

（3）三分支结构。

与高级语言类似，三分支以及更多分支结构的程序可以通过嵌套的方式实现。此外，在 51 单片机的汇编语言程序中，利用 CJNE 指令也可以很方便地实现。

假设要根据 R0 中数据的正负，分别将常数 0、＋1 和－1 存入累加器 A，可以编写如下程序：

```
        CJNE   R0,#0,NZERO     ; R0 = 0?
        MOV    A,#0            ; 是,则将 0 存入 A
        SJMP   DONE
NZERO:  CJNE   R0,#80H,NEXT    ; 否则,继续判断正负
NEXT:   JC     PLUS            ; 正数,则跳转
        MOV    A,#(-1)         ; 否则,将 -1 存入 A
        SJMP   DONE
PLUS:   MOV    A,#1            ; 将 +1 存入 A
DONE:   …
```

上述程序的流程图如图 3-17 所示。下面对上述程序做一些解释说明。

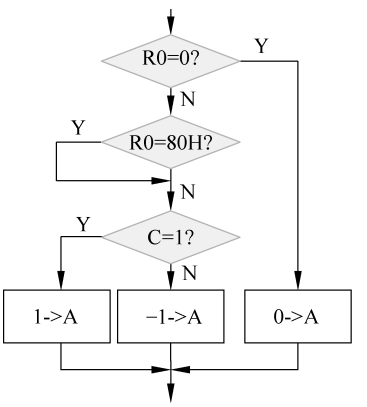


图 3-17　三分支程序

- 第一条 CJNE 指令用于判断 R0 中的数据是否为 0。如果为 0，则不跳转，继续执行第二条 MOV 指令，将常数 0 存入 A，之后利用 SIMP 指令直接跳转到 DONE 标号所在的指令继续执行程序中的其他指令。如果 R0 中的常数不为 0，则执行第一条 CJNE 指令后将跳转到标号为 NZERO 的指令。
- 第二条 CJNE 指令是在上述 R0≠0 的前提下，将 R0 中的数据继续与常数 80H 相比较，判断 R0 中的数据是否等于 80H。如果不是，则跳转到标号为 NEXT 的指令；如果相等，则不跳转而顺序执行该条 CJNE 指令后面的指令。由此可见，不管 R0 中的数据是否等于 80H，执行该条 CJNE 指令后都将继续执行标号 NEXT 所在的指令。显然，该条 CJNE 指令没有实现分支。

- 实际上，第二条 CJNE 指令的主要作用是根据两个操作数的相对大小设置 C 标志位，以便接下来利用 JC 指令检测 C 标志位，以判断两个数的相对大小，并实现两路分支。

例如，假设 R0 中存放的有符号数为＋10＝0AH，则将其与 80H 相减，显然 C＝1。因此执行 JC 指令后将跳转到标号为 PLUS 的指令，将常数 1 存入累加器 A。如果 R0 中存放的有符号数为负数，例如－10，则执行 CJNE 指令时，将－10 的补码即 0F6H 与 80H 相减。显然此时没有借位，因此 C＝0，执行 JC 指令后将不跳转，而是顺序执行后面的指令，将常数－1（即其补码 0FFH）存入累加器 A。

- 程序中的关键数据 80H 是这样确定的。分析题目可知 R0 中存放的原始数据是有符号数，而在汇编语言程序中，默认的有符号数用补码表示，并且负数的补码最高位一定为 1，正数的补码最高位一定为 0，因此负数一定为 80H～0FFH，而正数为 01H～7FH。由此可知，将 R0 中的数据与 80H 相减，根据是否有借位即可区分正负数。

2. 循环程序

循环程序的基本结构和功能可以归纳为如下两种典型的情况：

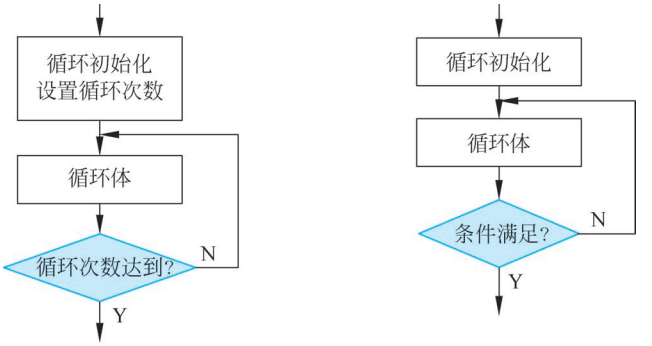
（1）重复执行一个程序段若干次，当重复次数达到后，退出循环程序并继续执行后面的程序。在编写这种循环程序时，事先能够确定程序段重复执行的次数，称为**计数控制循环**。

（2）事先无法确定程序需要重复执行多少次，但可以确定当满足（或不满足）一定的条件才继续循环，当条件不满足（或满足）时退出循环。这种循环程序称为**条件控制循环**。

上述两种循环结构的程序流程图如图 3-18 所示。在汇编语言程序中，根据控制循环的次数或条件，合理选用各种转移指令，很容易构成上述两种循环程序。一般的用法可以概括为：

（1）对计数控制循环，在循环开始前将循环次数保存到 Rn 或某个内部 RAM 单元。每次循环，用 DJNZ 指令将循环计数变量减 1，并实现跳转循环。

（2）对条件控制循环，根据题目要求的功能确定合适的条件，再选用相应的指令设置和修改条件，并选用合适的条件转移指令（JZ/JNZ、CJNE 等）以控制继续或退出循环。



(a) 计数控制循环　　(b) 条件控制循环

图 3-18　基本循环结构的程序流程图

在动手实践 3-2 的程序 **p3_2.asm** 的延时子程序中，一共有 3 层计数控制循环。在每层循环的开始，将各层循环所需的循环次数分别存入 R5、R6 和 R7，这 3 个工作寄存器相当于计数变量。在每层循环中都用 DJNZ 指令对该层循环所用的计数变量实现递减操作，同时判断是否减到零。如果没有减到零，表示循环次数还未到，则继续循环；否则退出循环。

在延时子程序中，每条指令后面的注释给出了执行该指令所需的机器周期数，据此可以确定该子程序执行完毕所需的时间，也就是延时的时间。对于最内层用 R6 作为计数变量的循环，循环体中只有一条 DJNZ 指令。每执行一次需要 2 个机器周期。由于 R6 设初值为 200，因此该循环将重复执行 200 次，共需 200×2 个机器周期。

同理，在第二层用 R7 作为计数变量的循环中，有一条 MOV 指令和两条 DJNZ 指令，执行该层循环所需的机器周期数为 200×(1+200×2+2)。最外层用 R5 控制的循环共重复 13 次，则执行 3 层循环一共需要的机器周期数为

$$13\times[1+200\times(1+200\times2+2)+2]=1047839$$

再考虑到子程序最后执行 RET 指令和主程序执行 ACALL 指令所需的时间，则在主程序中调用执行一次该子程序，总的机器周期数为

$$1047839+2+2=1047843$$

假设单片机系统的时钟脉冲频率 12 MHz，则机器周期为 12/12 MHz=1 μs，因此执行该延时子程序一共需要近似 1 s 的时间。

3. 死循环

所谓**死循环**(Endless Loop)，就是程序一旦启动运行后，就一直重复运行，直到强制退出应用程序或者系统关机。51 单片机系统的应用程序一般都是死循环，因为系统一旦启动运行后，就希望一直运行，重复实现相同的操作，例如重复不断地检测并调节当前环境温度。

在**动手实践 3-2** 中，每隔一段时间，重复不断地让 LED 在亮/灭状态之间切换。在**动手实践 3-3** 中，重复不断地检测 8 个开关的状态。在程序中，用无条件转移指令可以很方便地实现死循环。例如，在上述两个动手实践中，主程序的最后都是一条 SJMP 指令。当主程序中的具体功能操作完成后，执行该条无条件转移指令，又返回到主程序的开始或者标号为 LP 的指令重复执行。

请读者思考以下几个问题：

- **动手实践 3-1** 的主程序为什么没有用死循环？
- 在动手实践 3-4 的程序 **p3_4_1.asm** 中，死循环中包括哪些语句？
- 在程序 **3_4_2.asm** 的主程序中，有死循环吗？

3.3.5 子程序和堆栈

在程序中，如果有一个程序段实现的功能相对独立，或者在同一个或多个程序中需要反复使用，则一般将该程序段定义为**子程序**(Subroutine)。例如，在动手实践 3-2 的程序 **p3_2.asm** 中，将实现延时功能的代码定义为延时子程序 DELAY。

在51单片机的汇编语言程序中，子程序和主程序位于同一个程序文件中，一般放在主程序后面，但必须在END语句之前。此外，在子程序的前面也可以用ORG指定子程序在ROM中的存放位置。

1. 子程序的调用与返回

在汇编语言程序的主程序中，通过执行ACALL或LCALL指令即可调用子程序。ACALL称为**绝对调用指令**，LCALL称为**长调用指令**。这两条指令的功能和用法分别与AJMP和LJMP指令类似，一般将需要调用的子程序名字直接作为指令的操作数。所谓子程序的名字，也就是子程序中第一条指令的标号。

从底层实现原理的角度看，执行ACALL或LCALL指令调用子程序时，实现的功能是将子程序中第一条指令的地址（称为**子程序的入口地址**）送到PC，从而跳转到子程序，连续执行子程序中的指令。

子程序调用与无条件转移都能控制程序执行流程的切换。二者的主要区别在于：执行无条件转移指令跳转到目标位置后不再返回原位置；而子程序调用跳转到子程序，子程序执行完毕后要通过执行**子程序返回指令** RET返回主程序。该指令必须放在子程序的最后。

从子程序返回到主程序，返回的位置是主程序刚才被打断的位置，也就是主程序中LCALL或ACALL指令后面一条指令在ROM中存放的单元，该单元的地址称为**断点**（Breakpoint）。为了能够从子程序正确返回，必须在进入子程序之前将断点保存到51单片机中的适当位置。

子程序的调用与返回过程示意图如图3-19所示，其中LCALL指令共3字节，后面2字节0300H即为子程序SUB0的入口地址。执行程序时，当CPU取出该条指令后，PC指向下一个单元，该单元的地址即为断点。在执行LCALL指令时，将执行如下两个操作：

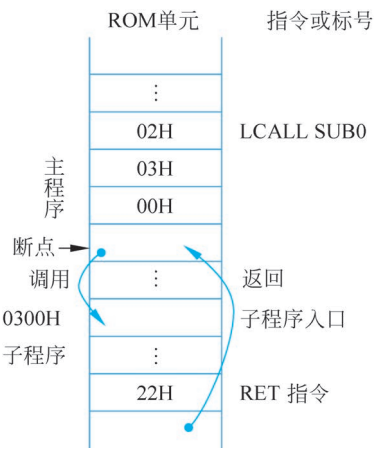


图3-19　子程序的调用与返回过程示意图

（1）将断点保存到指定位置（堆栈）；

（2）将子程序的入口地址0300H赋给PC，即跳转到子程序。

当执行到子程序中最后的RET指令时，再将断点从堆栈中取出重新赋给PC，从而返回断点位置继续运行主程序。

2. 堆栈及其应用

在51单片机中，**堆栈**（Stack）是一段特殊的内部RAM区域。堆栈的特殊性在于其访问方式与普通的内部RAM单元不同。普通的内部RAM单元可以实现随机访问，在任何时刻给定一个单元地址，都可以读取或写入期望的数据。但是，堆栈中的单元只能按照“**先进后出**”或“**后进先出**”的原则进行访问。

（1）堆栈单元的访问。

对堆栈的访问有两种基本操作，即入栈和出栈。所谓**入栈**，就是将数据写入堆栈单元；所谓**出栈**，就是从堆栈单元读出数据。在 51 单片机中，入栈和出栈操作分别用专门的 PUSH 和 POP 指令实现。这两条指令都只需要一个操作数，并且只能采用直接寻址。

例如，要将累加器 A 中的字节数据入栈，可以用如下指令：

```
PUSH   ACC
```

注意，操作数不能写为 A。这里的 ACC 汇编后将用累加器对应的特殊功能寄存器单元地址 0E0H 替换。

类似地，如果希望从堆栈单元中取出一个数据保存到工作寄存器 R0 中，可以用如下指令：

```
POP   00H
```

注意，操作数不能直接写为 R0，即不能采用寄存器寻址，必须采用直接寻址。

（2）堆栈指针。

在 51 单片机中，**堆栈指针**（Stack Pointer，SP）是一个专用的 8 位寄存器，在运行过程中，SP 中保存的是当前堆栈中栈顶单元的地址，或者说，SP 指向当前栈顶。

所谓**栈顶**，也就是在程序执行当前最后一次堆栈操作后，堆栈中最后一个数据所存放的单元。在程序运行过程中，可能会执行多次入栈或出栈操作，将多个数据存入堆栈单元或者从堆栈单元中取出，从而使得堆栈中所存放数据的个数在不断变化，栈顶将不断上下浮动。

为了使 SP 在任何时刻都指向当前栈顶，每执行一次入栈操作，SP 先递增 1，然后将 1 字节数据推入堆栈。反之，每执行一次出栈操作，将 1 字节数据从堆栈中弹出后，SP 再递减 1。

51 单片机系统刚启动或者复位后，SP 中的内容初始化为 07H，这意味着当前栈顶为 07H 单元。执行第一次入栈操作时，数据将保存到 SP＋1 指向的 08H 单元，这意味着系统默认将地址从 08H 开始的内部 RAM 单元作为堆栈。考虑到内部 RAM 开始的 32 个单元是工作寄存器区，一般在需要用到堆栈时，在主程序最前面用如下指令为 SP 重新赋值，以便将堆栈定位到其他内部 RAM 区域。例如，将该指令中的操作数设为 60H，则会将堆栈定位到一般 RAM 区中 61H 开始的单元。

```
MOV   SP,#dat
```

（3）堆栈的应用。

在 51 单片机系统中，子程序断点的保存和恢复都是利用堆栈实现的。在执行 ACALL 或 LCALL 指令调用子程序时，自动将断点推入堆栈。执行 RET 指令从子程序返回时，自动从堆栈将断点出栈存入 PC，从而正确返回主程序。由于断点指的是指令代码在 ROM 中

存放的单元地址，因此断点都是 16 位二进制数据，断点入栈和出栈都需要分别连续执行两次，执行后 SP 将分别加 2 或减 2。

此外，利用堆栈特殊的访问规则，还可以实现一些特殊的功能。例如，将内部 RAM 中地址为 30H 和 31H 的两个单元中的数据交换，可以用如下 5 条指令实现：

```
MOV   SP,#60H
PUSH  30H
PUSH  31H
POP   30H
POP   31H
```

假设 30H 和 31H 两个单元中原来存放的数据分别为 x 和 y，则执行完上述两条入栈操作指令后，堆栈中各单元的分配情况如图 3-20(a)所示，并且 SP＝60H＋2＝62H。

接下来执行第一条 POP 指令时，将当前栈顶单元中的数据 y 出栈，存入该指令中指定的 30H 单元，SP＝62H－1＝61H，此时堆栈单元分配情况如图 3-20(b)所示。再执行最后一条 POP 指令，将 SP 指向的当前栈顶即 61H 单元中的数据 x 出栈，存入 31H，并且 SP＝61H－1＝60H，此时堆栈单元分配情况如图 3-20(c)所示。

由此可见，上述两次出栈操作完成后，30H 和 31H 单元中分别存入的是 y 和 x，从而实现了两个单元中数据的交换。

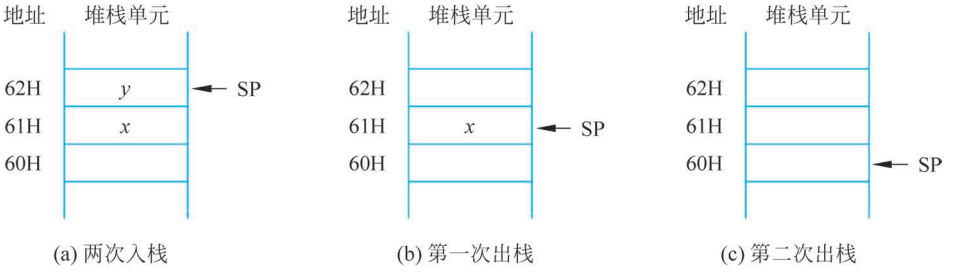


图 3-20　利用堆栈实现两个 RAM 单元中数据的交换

3.3.6　中断的基本概念及外部中断

由于计算机内部或者外部的原因(称为随机事件)，使 CPU 暂停当前正在执行的程序，转而执行预先安排好的服务程序，对该事件进行判断处理，处理完后再继续执行原来被打断的程序。这一过程称为**中断**(Interrupt)。实现中断过程管理和控制的所有硬件和软件统称为**中断系统**(Interrupt System)。

1. 中断源

在单片机系统中，产生中断随机事件的来源，称为**中断源**。中断源产生的随机事件称为**中断请求**。例如，按钮电路可以视为一个中断源。每次按钮按下时，按钮电路输出一个负脉冲，即由原来的高电平跳变到低电平，就是一次中断请求。

中断请求可以来自单片机内部电路，也可以是由外部设备电路产生而送入单片机内部

的。由外部电路通过 P3.2 或 P3.3 引脚送入单片机的中断请求称为**外部中断**。这两个引脚用作中断请求信号输入引脚而不是普通的并口引脚时，引脚名重新表示为 $\overline{\text{INT0}}$ 和 $\overline{\text{INT1}}$，称为 P3 口的**第二功能**。由内部的定时/计数器、串口电路等发出的中断，称为单片机的**内部中断**。这些内部中断不需要占用单片机的引脚，而是在单片机内部产生，直接送到 CPU。

这里首先介绍外部中断，定时/计数器和串口等内部中断将在后续章节介绍。

2. 外部中断请求的引入

当中断请求到来时，意味着外部发生了特定的某种事件。例如用户按动了按钮，希望点亮某个 LED 或者报警。在运行过程中，单片机在每个机器周期都将检测 P3.2 或 P3.3 引脚上电平的变化。一旦检测到引脚上出现了特定的电平状态，就认为外部中断源送来了中断请求。

在 51 单片机中，送入 51 单片机的可以是中断请求信号输入引脚上的高/低电平，也可以是电平的跳变（正跳变或者负跳变）。这两种情况称为外部中断请求的**触发方式**。

在 51 单片机内部 RAM 的特殊功能寄存器区，有一个**定时/计数器控制（Timer/Counter Control）寄存器 TCON**。该特殊功能寄存器的地址为 88H，因此可以位寻址，各位的名称如图 3-21 所示。其中，IT0 和 IT1 就是用于设置两个外部中断请求的触发方式。当这两位设置为 0 时，指定对应的外部中断为电平触发方式；当 IT0 或 IT1 设置为 1 时，指定为边沿触发方式。

D7	D6	D5	D4	D3	D2	D1	D0
TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

图 3-21　TCON 寄存器

对于**电平触发**方式，CPU 在一个机器周期内检测到 P3.2 或 P3.3 引脚上为低电平，就认为接收到一个中断请求。对于**边沿触发**方式，51 单片机将在相邻两个机器周期内分别检测一次 P3.2 或 P3.3 引脚的状态，一旦检测到这两个引脚上出现负跳变（从高电平跳变到低电平），CPU 就认为外部电路送来了一个中断请求。51 单片机一旦检测到 P3.2 或 P3.3 引脚有外部中断请求到来，将立即使 TCON 中的 IE0 或 IE1 位置位，以便将当前中断请求记录下来，进一步通知 CPU 并等待 CPU 的处理。

在**动手实践 3-4** 中，按钮电路的中断请求通过 P3.2 引入，因此是外部中断 $\overline{\text{INT0}}$。利用如下位操作指令：

```
SETB   IT0
```

将 TCON 中的 IT0 位设置为 1，则当每次按钮按下时，按钮电路产生一次负跳变，从而向 CPU 发出一次 $\overline{\text{INT0}}$ 中断请求。

3. 外部中断请求的撤除

对于边沿触发方式，51 单片机每检测到引脚上的一次负跳变，就将 IE0 或 IE1 位置位。

一旦 CPU 接收并准备处理该中断请求后，内部硬件电路会自动将 IE0 或 IE1 位复位，称为**中断请求的撤除**。之后只有当用户再次按下按钮，才能重新将 IE0 或 IE1 位置位，向 CPU 发出一次新的中断请求。

正常情况下，在运行过程中，用户每按下一次按钮，51 单片机应该只接收到一次中断请求，做一次相应的处理操作。上述撤除操作是必需的。但是对电平触发方式，CPU 接收并响应一次中断请求后，IE0 或 IE1 位不会自动复位，从而将导致每按动一次按钮，CPU 会重复接收到多次中断请求的情况。为此，在实际系统中必须用另外的专门电路使 P3.2 或 P3.3 引脚变为高电平。以便等到用户再次按下按钮，重新将引脚变为低电平，才能向 CPU 发出一次新的中断请求。

图 3-22 是一种典型中断请求的撤除电路。以按钮为例。当用户按下按钮时，按钮电路送来一个外部中断请求信号，使 D 触发器的 Q 端变为低电平，向 51 单片机发出一次中断请求。

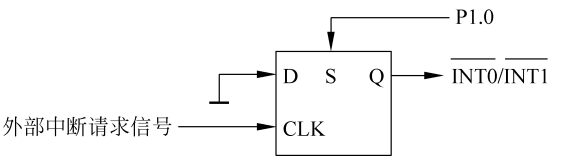


图 3-22　中断请求撤除电路

51 单片机接收到该中断请求后，在响应和处理时先执行如下指令：

```
SETB  P1.0
```

由 P1.0 引脚送出一个高电平，通过 D 触发器的 S 端使 Q 端输出强制变为高电平，从而撤除当前中断请求。之后，再执行如下指令：

```
CLR  P1.0
```

或者

```
CPL    P1.0
```

将 P1.0 输出变为低电平。等到当用户再次按下按钮时，D 触发器的 Q 端又可再次发出中断请求。

4. 中断响应

CPU 一旦检测接收到中断请求，就应该及时予以处理。CPU 暂停当前正在进行的处理和操作，转而处理中断事件的过程，称为**中断响应**。

(1) 中断响应的条件。

在系统运行过程中，并不是每个中断请求到来后都能及时得到 CPU 的响应和处理。也就是说，CPU 接收到中断请求后，该中断请求要得到 CPU 的响应和处理，必须满足一定的条件。例如，由于中断请求是由外部电路送来的，而外部电路和单片机的工作是相对独立

的，因此对 CPU 来说，接收到中断请求的时刻是随机的。当中断请求到来时，CPU 可能正在执行某条指令。显然，必须要等到当前指令执行完毕后才能响应中断请求。

此外，当外部中断请求到来时，只是将 TCON 中的 IE0 或 IE1 位置位。中断请求必须进一步送到 CPU，CPU 才能响应和处理。在程序中，必须在中断请求到来前（一般在主程序一开始）执行如下指令：

```
SETB  EX0(EX1)
SETB  EA
```

CPU 才能真正接收到中断请求。

在上述两条指令中，EX0、EX1 和 EA 是**中断允许（Interrupt Enable，IE）寄存器**（地址为 0A8H）中的两位。该寄存器各位的含义如图 3-23 所示。

D7	D6	D5	D4	D3	D2	D1	D0
EA		ET2	ES	ET1	EX1	ET0	EX0

图 3-23　IE 寄存器各位的含义

在上述两条指令中，第一条指令将 IE 中的 EX0 或 EX1 设为 1，表示运行 CPU 响应外部中断 0 或 1。第二条指令设置 EA 位为 1，表示开放所有的中断，这相当于一个总开关。

当 EA＝1 时，如果 EX0＝1，则允许 CPU 响应和处理外部中断 0，称为**开中断**；否则，如果用 CLR 指令将 EA 或 EX0 清 0，外部中断 0 的中断请求将无法送到 CPU，则 CPU 也就不会响应，称为**关中断**。

（2）中断响应。

在中断响应过程中，将由单片机内部的硬件电路自动实现如下功能：

- 保护断点，也就是将 PC 的内容推入堆栈保存起来；
- 对边沿触发方式，将 TCON 中的 IE0 或 IE1 位复位；
- 将中断服务程序的入口地址送入 PC，从而转入相应的中断服务程序。

由此可见，中断响应的过程与子程序调用是类似的，在中断响应的过程中，也要实现程序执行流程的切换。

51 单片机一共有 2 个外部中断、2 个或 3 个定时/计数器中断、一个串口中断，这些中断源的中断服务程序必须分别放在 ROM 中指定的单元，这些单元构成**中断向量表**，如表 3-1 所示。

如果中断服务程序实现的处理操作比较简单，长度不超过 8 字节，可以直接将中断服务程序代码放在表中对应的 ROM 单元。但是很多情况下，各中断源的中断服务程序都远远超过 8 字节。为此，一般在这些单元中存放一条长度只有 2 字节或 3 字节的转移指令 AJMP 或 LJMP，而将中断服务程序真正的入口地址（一般为标号）作为这两条转移指令的操作数。

表 3-1　ROM 中的中断向量表

ROM 单元地址	对应的中断源
0003H	外部中断 0
000BH	定时/计数器 0 中断
0013H	外部中断 1
001BH	定时/计数器 1 中断
0023H	串口中断
002BH	定时/计数器 2 中断(仅 52 子系列有)

当 CPU 响应某个中断请求时,将根据接收到的中断请示对应的是哪个中断源,自动跳转到表中对应的单元,执行其中存放的中断服务程序,或者执行转移指令跳转到真正的中断服务程序。

在动手实践 3-4 的程序 **p3_4_2.asm** 中,利用中断技术实现按钮状态的检测,用到了外部中断 0,因此必须在地址为 0003H 开始的单元安排一条无转移指令 AJMP、LJMP 或 SJMP,跳转到按钮的中断服务程序,相关的程序代码如下:

```
ORG   0003H          ; 指定 AJMP 指令存放的单元
AJMP  INT0_DEL       ; 跳转到真正的中断服务程序
```

5. 中断服务程序

一般来说,一个单片机系统会有很多中断源和中断请求,而每个中断源需要 CPU 实现的操作和服务处理、发出中断请求后要实现的功能是各不相同的。为每个中断源分别编写,实现中断服务处理的程序称为**中断服务子程序**,或者简称为**中断服务程序**。

中断服务程序类似于子程序。但是中断服务程序中最后执行的指令必须是中断返回指令 RETI,其作用相当于普通子程序中的 RET 指令,执行后将从堆栈中取出断点,从而返回中断请求发生前 CPU 正在执行的主程序。

需要强调的是,从程序本身来看,主程序与中断服务程序之间没有关系,是相互独立的。中断服务程序什么时候得到执行,在编写程序时是无法确定的,完全取决于运行时中断请求什么时候到来。基于此,在画程序流程图时,主程序和中断服务程序的流程图必须分开绘制。在分析程序时,必须考虑到上述中断响应和程序流程的切换过程,从而将主程序和中断服务程序联系起来。

例如,在动手实践 3-4 的程序 **p3_4_2.asm** 中,当执行到主程序最后的死循环时,CPU 不断重复执行 SJMP $ 指令,等待按钮电路发来中断请求。一旦用户按下按钮,就立即向 CPU 发出中断请求。如果条件允许,则 CPU 响应中断,暂时停止主程序的执行,通过上述中断响应过程跳转到按钮的中断服务程序。中断服务处理完毕后,再返回到主程序中的死循环,继续等待下一次按钮按下。如果在系统运行过程中,用户始终未按下按钮,则中断服务程序将永远不会执行。图 3-24 给出了程序 p3_4_2.asm 的流程图。

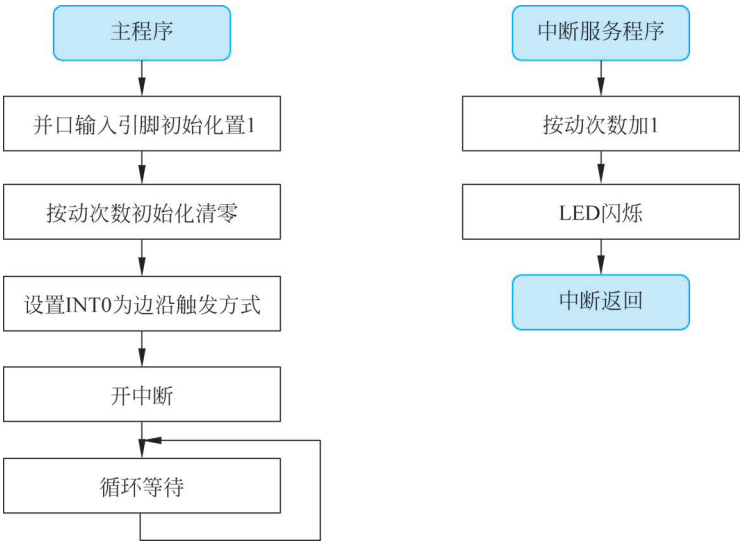


图 3-24　程序 p3_4_2.asm 的流程图

3.4　牛气冲天——实战进阶

前面通过几个案例介绍了与并口基本数据输入/输出功能和 51 单片机外部中断相关的基本概念及典型用法，本节将继续介绍上述基本概念的综合应用及知识拓展。

3.4.1　外部中断源的扩展

实际系统中可能所需的外部中断远不止一个或两个。为了实现外部中断源的扩展，即能够通过 51 单片机的两个外部中断请求引脚引入更多的外部中断请求，从电路上说，可以将多个中断请求通过基本的门电路组合合并为一个中断请求，然后通过同一个引脚送入 51 单片机。

由于每个中断请求要求得到的中断服务和处理各不相同，因此在程序上还需要进一步区分当前中断请求具体是由哪个中断源（例如哪个按钮）引起的。为此，可以将各中断请求信号再分别通过不同的并口引脚送入 51 单片机。

在程序中，一旦检测到有外部中断请求时，再从某个并口读入各按钮的中断请求信号，并依次判断相应的各位是否为低电平。当检测到某位为低电平时，即可确定是由对应的按钮发出的中断请求，据此继续做其他操作。

下面通过实际案例体会中断源扩展的基本方法。

动手实践 3-5：多个按钮状态的检测

本案例的原理图如图 3-25 所示（参见 Proteus 工程文件 **ex3_5.pdsprj**）。电路中共有 4 个按钮，要求采用中断方式实现各按钮状态的检测。当按下某个按钮时，用 8 个 LED 显示按

钮序号对应的 ASCII 码。例如，按下按钮 B1 时，其序号 1 的 ASCII 码为 31H＝00110001B，则从左往右第 3、4、8 个 LED 点亮，其他 LED 熄灭。

该电路的工作原理是：只要有一个按钮按下，则与门 U2 输出端产生一个负跳变，向 51 单片机发出一个中断请求。一旦有按钮按下，再通过程序继续检测和识别具体是哪个按钮按下，据此实现不同的操作。

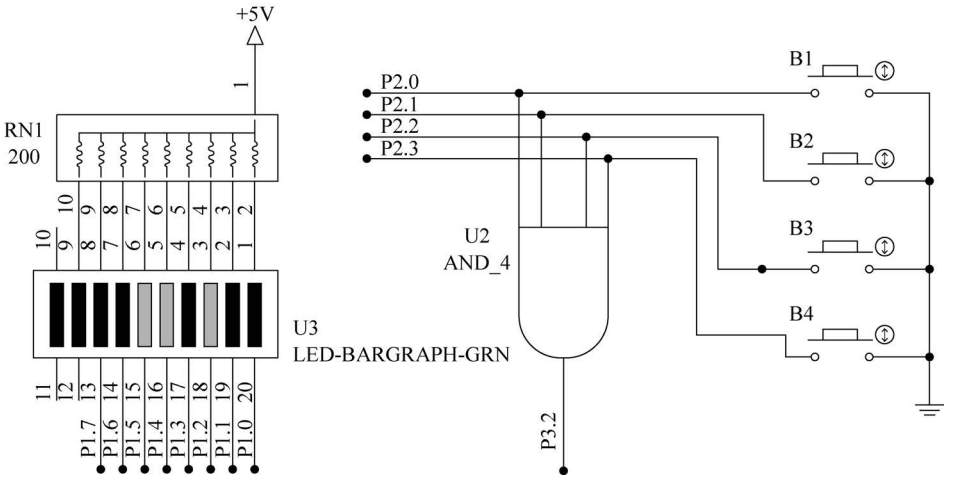


图 3-25　多个按钮状态的检测

本案例完整的程序代码如下（参见文件 **p3_5. asm**）：

```
; 多个按钮状态的检测——中断方式
BT    BIT   P3.2          ; 将按钮状态输入引脚 P3.2 定义为位变量 BT
LED   EQU   P1            ; 将控制 LED 的 P1 口定义为符号常量 LED
      ORG   0000H
      AJMP  MAIN
      ORG   0003H
      AJMP  IDEL          ; 外部中断 0 服务程序入口
;=============================================================
;主程序
;=============================================================
      ORG   0100H
MAIN: MOV   SP,#60H       ; 设置堆栈指针
      MOV   P2,#0FH       ; P2 口低 4 位输出高电平
      SETB  BT            ; P3.2 先输出高电平
      MOV   LED,#0FFH     ; 初始熄灭所有 LED
      SETB  IT0           ; 设置外部中断 0,边沿触发方式
      SETB  EX0           ; 开中断
      SETB  EA
      SJMP  $             ; 死循环,等待按钮按下
;=============================================================
;外部中断 0 中断服务程序
;=============================================================
IDEL: JB    P2.0,NEXT1    ; 按钮 B1 按下?
```

```
        MOV     A,#31H          ;是,将1的ASCII码31H存入A
        SJMP    DONE            ;跳转
NEXT1:  JB      P2.1,NEXT2      ;按钮B2按下?
        MOV     A,#32H          ;是,将2的ASCII码32H存入A
        SJMP    DONE
NEXT2:  JB      P2.2,NEXT3      ;按钮B2按下?
        MOV     A,#33H          ;是,将3的ASCII码32H存入A
        SJMP    DONE
NEXT3:  MOV     A,#34H          ;B4按下,将4的ASCII码34H存入A
DONE:   CPL     A
        MOV     LED,A           ;输出控制LED显示ASCII码
        RETI                    ;中断返回
        END
```

在上述程序的主程序中，首先做了如下一系列初始化操作：

- 设置堆栈指针；
- 由于 P2 口的低 4 位和 P3.2 引脚用作输入口，因此需要先将这些引脚置 1；
- 初始熄灭所有 LED；
- 与中断相关的初始化设置（设置外部中断的触发方式、开中断）。

在接下来不断重复执行 SJMP 指令的过程中，等待按钮按下。一旦有按钮按下，则 51 单片机响应中断请求，转到中断服务程序。

在中断服务程序中，主要是利用 3 条 JB 指令依次检测 P2.0～P2.3 是否为高电平。如果是，则表示相应引脚连接的按钮没有按下，继续检测下一个按钮。否则，相应按钮按下，则将相应按钮序号 1～4 的 ASCII 码存入累加器 A，最后将其取反后由 P1 口输出控制 LED 的亮灭。

3.4.2 中断优先级的简单理解

在**动手实践 3-5** 的中断服务程序中，对各按钮检测的顺序依次是 P2.0～P2.3。当检测到 P2.0＝0 时，意味着 P2.0 引脚连接的按钮 B1 有中断请求，此时跳转到标号为 DONE 的语句，执行对应的操作后返回主程序，不再执行循环中后面的其他 JB 指令。

由此可见，只有检测到按钮 B1 没有按下时，才能执行检测按钮 B2 是否按下的操作；只有按钮 B1 和 B2 都没有按下时，才能继续检测按钮 B3 是否按下，……。

上述执行过程意味着，只有当按钮 B1 没有按下时，按钮 B2 的中断请求才能得到响应。同理，只有按钮 B2 没有按下时，才可能响应按钮 B3 的中断请求，……。因此说按钮 B1 的中断优先级高于按钮 B2，按钮 B2 的中断优先级高于按钮 B3，……。显然，上述各条 JB 指令检测 P2 口各位的顺序可以任意规定。对 P2 口各位的检测顺序决定了各按钮中断的优先级别。这就是**中断优先级**（Priority of Interruption）表示的含义和实现的功能。

本章小结

本章介绍了 51 单片机内部集成的并行 I/O 接口及其连接输出设备时的基本用法，并初步认识了 51 单片机的外部中断及其应用。

1. 并口及其基本用法

(1) 51单片机内部集成了4个并口，可以分别连接4个并行外部设备(例如8个LED、8个开关或按钮等)。硬件连接时，P0口必须外接上拉电阻，其他3个并口不需要上拉电阻。

(2) 51单片机通过4个并口可以实现字节操作，也可以实现位操作。在程序中用MOV指令既可通过并口实现字节数据的输入或输出，也可以用位操作指令实现位操作，控制指定的某个并口引脚输出高/低电平，或者读取某个给定并口引脚的高/低电平状态。

(3) 当并口用于连接输入外部设备时，在从并口读取数据前，必须用指令先使所用的并口或引脚输出高电平，否则外部电路无法使并口引脚的高/低电平正确变化。

(4) LED、开关和按钮是在51单片机应用系统中常用的开关器件，利用并口可以很方便地对其实时控制，这些器件与51单片机的连接电路大都是典型的标准电路。设计LED电路时，主要考虑其限流电阻的问题；设计开关和按钮电路时，主要考虑其消抖问题。

2. 单片机的指令系统

在本章所给的各动手实践案例中，编制的汇编语言程序主要用到51单片机指令系统中的位操作指令、条件转移指令、子程序的调用与返回指令等。

(1) 位操作指令是51单片机中功能强大、应用最频繁的指令。利用位操作指令可以设置指定并口引脚输出的高/低电平，检测指定并口引脚的电平状态，从而实现开关量的输入或输出。

(2) 利用条件转移指令可以很方便地实现程序的分支和循环，对开关量的输入和状态检测也大都利用条件转移指令和位转移指令实现。

(3) 在汇编语言程序中，利用各种条件转移指令可以很方便地实现分支和循环程序。编写程序时，关键是确定合适的转移条件，正确选用相应的条件转移指令。

(4) 与高级语言类似，在汇编语言程序中，通常将实现相同功能的程序段，或者功能相对独立的程序段单独编写为子程序，在主程序中需要的地方直接调用。在汇编语言程序中，子程序的调用和返回都有专门的指令实现，因此可以很直观地体会到子程序的调用和返回过程。

3. 单片机的中断系统与外部中断

中断是任何微机系统中都具备的一项重要技术，利用中断可以使CPU和外部设备同步工作，提高CPU的工作效率，并能使CPU对外部的某些特殊事件作出及时的反应。外部中断的一个典型应用就是实现开关通断状态和按钮动作的实时检测。

(1) 51单片机中的所有中断请求分为两大类，即内部中断和外部中断。所有的外部中断都是通过P3口的P3.2和P3.3引脚引入。

(2) 在硬件设计时，只需要将外部中断源电路连接到P3.2或P3.3引脚。当外部中断事件发生时，通过外部中断源电路使这两个引脚变为低电平或者出现负跳变，即可向CPU发出中断请求。

(3) 要使用外部中断，在程序中，必须首先设置外部中断的触发方式，之后开中断以便CPU接收到中断请求。这两项操作都是在主程序中的一开始，通过对特殊功能寄存器

TCON 中相应的位进行位操作设置实现的。之后，主程序中通过不断重复执行循环程序实现其他操作，同时等待中断请求的到来并响应和处理中断。

（4）当 CPU 接收到外部中断请求后，在响应中断的过程中，将自动跳转到地址为 0003H 或 0013H 的 ROM 单元。大多数情况下，在这些单元中存放一条条件转移指令，以便跳转到相应的中断服务程序。

（5）在汇编语言程序中，中断服务程序一般与主程序放在同一个程序文件中，一般放在主程序的后面，END 语句之前。中断服务程序的最后必须是一条 RETI 指令。

（6）51 单片机只有两个引脚 P3.2 和 P3.3 能够引入外部中断。如果一个系统中的外部中断比较多，可以进行外部中断源的扩展。扩展时需要同时进行相应的外部电路和程序设计，这些设计都有典型的方法供参考。

思考练习

3-1　填空题

（1）在 51 单片机中，具有第二功能的并口是_____。

（2）在 51 单片机中，存在漏极开路问题的并口是_____口，在用作普通并口时，必须在 51 单片机外部将该并口的各引脚接_____。

（3）51 单片机复位时，所有并口引脚都输出_____电平。

（4）为了使 51 单片机的 P1.7 引脚输出低电平，可以使用指令_____，为了使 P1 口的全部引脚输出高电平，可以用指令_____实现。

（5）已知 P3＝0FFH，则执行指令 CPL P3.7 后，P3.7 引脚输出_____电平，P3＝_____。

（6）在位逻辑运算指令中，目的操作数必须是_____。

（7）已知某程序中 JZ LP 指令汇编后的机器码为 60H、0F0H，存放在 ROM 中 0120H 和 0121H 单元，则当 A＝_____时，执行该指令将跳转到地址为_____的单元。

（8）已知某程序中 DJNZ R7，LP 指令汇编后的机器码为 0DFH、20H，存放在 ROM 中 011EH 和 011FH 单元，则当 R7＝2 和 1 时，执行该指令后将分别继续执行地址为_____和_____单元中存放的指令。

40H	0AH
41H	0BH
42H	00H
43H	02H
44H	04H

图 3-26　RAM 单元分配情况

（9）在 51 单片机中，子程序和中断服务子程序最后执行的指令分别是_____和_____。

（10）已知 SP＝4AH，则执行 LCALL 指令后，SP＝_____。

（11）已知 SP＝4AH，则执行 RET 指令后，SP＝_____。

（12）已知执行 ACALL DLY 指令后，内部 RAM 单元的分配情况如图 3-26 所示，并且当前栈顶为 43H，则该子程序调用指令存放在 ROM 中地址为_____开始的单元。

（13）在 51 单片机中，外部中断利用引脚_____或_____的

第二功能引入 51 单片机。

(14) 不管响应哪个中断请求，都必须用_____指令开中断，为了响应外部中断 0，还需要执行_____指令。

(15) 根据中断源所处的位置，51 单片机的中断分为_____和_____两种。

3-2　选择题

(1) 在 51 单片机的 4 个并口中，具有第二功能的是(　　)。

A. P0　　B. P1　　C. P2　　D. P3

(2) 当并口用作普通输入口连接一个输入设备时，在程序中应先做的操作是(　　)。

A. 输出低电平　　B. 输出高电平　　C. 输出　　D. 开中断

(3) 已知 P1.7 引脚上连接一个 LED 的阴极，则要点亮该 LED，不能执行的指令是(　　)。

A. CPL　P1.7　　B. MOV　P1，#7FH

C. CLR　P1.7　　D. ANL　P1，#7FH

(4) 已知 P2.0 引脚连接了一个 LED 的阳极，则要点亮该 LED，可以用(　　)指令实现。

A. CPL　P2.0　　B. CLR　P2.0

C. SETB 0A0H　　D. XRL　P2，#1

(5) 执行指令 CJNE A，#20H，NEXT 时，不做的操作是(　　)。

A. 将 A 中的数据与常数 20H 进行比较

B. 将 A 中的数据与 20H 相减，差存入 A

C. 根据 A 和常数 20H 的相对大小，设置 C 标志位

D. 如果 A≠20H，则跳转到 NEXT 标号处

(6) 已知 SP＝50H，则执行 3 次入栈操作和 1 次出栈操作后，SP＝(　　)。

A. 50H　　B. 51H　　C. 52H　　D. 53H

(7) 已知 SP＝50H，则执行子程序中的 RET 指令后，SP＝(　　)。

A. 50H　　B. 49H　　C. 4FH　　D. 4EH

(8) 在 51 单片机中，外部中断 1 的中断服务程序必须从 ROM 的(　　)单元开始存放。

A. 0000H　　B. 0003H　　C. 0013H　　D. 0100H

3-3　在图 3-9 中，已知 LED-RED 的参数如图 3-27 所示，计算限流电阻 $R1$ 的阻值。

3-4　简述 51 单片机中当并口用作输入口时的基本操作步骤。

3-5　在本章各动手实践案例中，都是用 51 单片机的并口引脚控制 LED 的阴极电平。如果用并口控制 LED 的阳极，而将阴极通过限流电阻接地，能否正常控制 LED 的亮灭？电路(包括元件参数)和控制程序该如何修改？

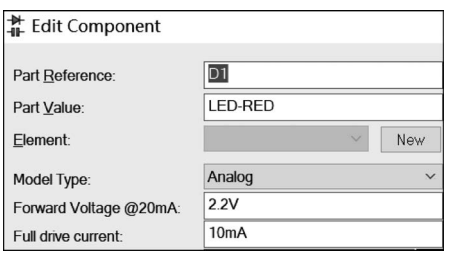


图 3-27　LED-RED 的参数

3-6　写出实现下列功能的指令(假设 LED 都采用共阳极接法)：

(1) 将 P1 口各引脚连接的所有 LED 亮/灭切换一次；

(2) 将 P1.7～P1.4 口连接的 4 个 LED 熄灭，另外 4 个引脚连接的 LED 保持原来的亮/灭状态；

(3) 要求用字节操作指令将 P1.7 引脚连接的 LED 点亮，而其余 7 个 LED 的亮/灭状态保持不变。

3-7　已知常数 1000 事先已经存入 R6(低字节)和 R7(高字节)，分别分析如下两个循环程序实现的功能。假设 51 单片机的时钟频率为 6 MHz，执行两个循环分别需要多少时间？

```
(1) NEXT:  CLR  C
           MOV  A,R6
           SUBB A,#1
           MOV  R6,A
           MOV  A,R7
           SUBB A,#0
           MOV  R7,A
           ORL  A,R6
           JNZ  NEXT
```

```
(2) LP:MOV  A,R7
       DEC  R7
       MOV  R2,0x06
       JNZ  CON
       DEC  R6
       CON: ORL  A,R2
       JZ   DONE
       SJMP LP
       DONE: …
```

3-8　已知 R6＝10H，R7＝20H，A＝00H，SP＝40H，执行如下程序段：

```
PUSH    ACC
PUSH    06H
PUSH    07H
POP     06H
POP     07H
```

(1) 当执行完第 3 条指令后，画出堆栈的存储单元分配图；

(2) 当执行完上述程序段后，A＝_____，R6＝_____，R7＝_____，SP＝_____。

3-9　如下汇编语言程序段实现的功能是：将内部 RAM 从地址为 30H 开始的单元中存放的数据进行累加。当累加和刚超过 200 时，点亮 LED(LED 阴极接 P1.0)。将程序补充完整，每处横线位置只能填一条指令。

```
        _____                    ; 初始熄灭 LED
    CLR  A
    MOV  R0,#30H
LP: _____                        ; 累加
     INC R0
        _____                    ; 累加和超过 200?
NEXT:_____                       ; 否，则继续
        _____                    ; 是，则点亮 LED
```

3-10　如下汇编语言程序段实现的功能是：将内部 RAM 中 30H～37H 单元的数据传送到 34H 开始的单元。将程序补充完整，每处横线位置只能填一条指令。

```
        MOV   R0,#37H                    ; 设置目的和源起始地址
        MOV   R1,#3BH
        MOV   R7,#8                      ; 设置数据个数
LP:     _____________________________    ; 传送 1 字节
        _____________________________
        _____________________________    ; 修改地址
        _____________________________
        _____________________________    ; 循环
```

综合设计

3-1　编写完整的汇编语言程序实现如下功能(要求加上必要的注释)：

(1) 子程序 DISP 实现如下功能：根据 R0 入口参数中的字节数据点亮或熄灭相应的 LED。例如，假设调用该子程序时 R0＝0F0H，则点亮 L1～L4 并熄灭 L5～L8，其中 8 个 LED 分别采用共阳极接法与 51 单片机的 P2.0～P2.7 相连接。

(2) 主程序中调用上述子程序，从 P1 口读入开关数据并点亮相应的 LED。

3-2　在 ROM 中从 0200H 单元开始存放有 20 个有符号数，编制完整的汇编语言程序实现如下功能(要求加上必要的注释)：

(1) 统计其中非负数的个数(提示：默认情况下，有符号数在 51 单片机中都用补码表示，负数补码的最高位为 1)。

(2) 将统计结果转换为 BCD 码，并用 8 个 LED 显示。假设 LED 采用共阳极接法与 P1 口相连接。

3-3　编写子程序实现一个通用延时子程序，延时的时间由子程序的入口参数确定。在主程序中调用该子程序，控制一个 LED 以不同的时间间隔进行闪烁。例如，亮 1 s、灭 2 s、亮 1 s、……。

3-4　在 51 单片机系统中，有两个 LED L1 和 L2 分别连接在 P1.0 和 P1.1 引脚，用两个按钮 B1 和 B2 分别作为两个外部中断 INT0 和 INT1，采用中断方式控制两个 LED 的闪烁。要求初始两个灯都点亮。当按动按钮 B1 时，L1 闪烁，直到按动按钮 B2；当按动按钮 B2 时，L2 闪烁，直到按动按钮 B1。

(1) 画出单片机与 LED 和按钮的连接电路。

(2) 编制主程序和中断服务程序。